Amendments to the Specification:

Please amend paragraph 0019 as follows:

After the OVERFLOW function finishes executing, the processor will pop off the return IP address and execute the instruction located at that address. In this example, the address pointed to by the integer value of X..X (length of pointer will depend on the system architecture) is probably not an instruction, and as a result, program 10 will probably crash. However array LARGE could have been intelligently loaded with input that places a meaningful address at the return IP location. After returning from the OVERFLOW function, the next instruction that will execute will be whatever instruction is stored in the address stored in the return IP location. If the attacker inserts instructions (*i.e.* code) into another location within the overrun buffer (*i.e.*, boxes 22-28 or beyond), then the attacker could also insure the return IP location (box ~~2~~26) then points to the location of his code and will be able to execute code of his own choice.

Please amend paragraph 0037 as follows:

Figure 3 is a chart showing ~~the~~ analysis times for stages of a pipeline, according to one embodiment of the invention.

Please amend paragraph 0043 as follows:

A certification system according to the present invention is designed such that vulnerability knowledge database (VKdb) 202 is designed as a distinct entity. VKdb 202 stores information about code vulnerabilities, that are used in Stage I of the pipeline. Stage I is an important stage of the pipeline because this is the stage where vulnerabilities are initially flagged. Accordingly, if a potential vulnerability ~~if~~ is not identified in Stage I, it will not be examined in later stages. VKdb ~~204~~ 202 is preferably designed with its own application programming interface (API such that other program modules can add and extract information from it. Also, a graphical user interface (GUI) may be included to simplify the process for industry professionals who populate the database with new vulnerabilities as they are identified.

Please amend paragraph 0080 as follows:

The analysis times (where computations were run on a Sun SPARC E450) for each stage of the pipeline are presented in chart 300, shown in FIG. 3. The analysis time for the grep utility are presented as a comparison. While the analysis time for the static analysis tool seems high, what is not presented is the graph of human time required for a code audit. The human time required to hand audit each individual vulnerability can be quite high, so every false positive eliminated from the list of vulnerabilities saves vast amounts of man-power resources. For a trade-off of an over-night computation the present invention decreased the number of false-positives presented to an auditor by fifty percent. Also not shown is the increase in accuracy of the analysis, because the results from the static analysis give a human a measure of which vulnerabilities are most likely to present a problem. The auditor can then give more attention to just those vulnerabilities and the subtle program interactions that can lead to a buffer overflow. The exploitable and the undetermined results will be passed on to the dynamic analysis module, and be further processed there.

Please amend the Abstract as follows (a copy of following on a separate sheet is also attached):

~~The present invention provides a methodology~~ A system and method for certifying software for essential and security-critical systems. The system and method provide a methodology and corresponding analysis engines increase the level of confidence that common vulnerabilities are not present in a particular application. A pipeline system consisting of independent modules which involve increasingly complex analysis is disclosed. The pipeline approach allows the user to reduce computation time by focusing resources on only those code segments which were not eliminated previously in the pipeline. ~~The first step of the pipeline consists of flagging all potential vulnerabilities contained in an extendible vulnerability knowledge database (VKdb). This stage then filters vulnerabilities based on context information and passes them to the second stage. The second stage performs complex static analysis on the vulnerabilities and passes the remaining ones to a dynamic analysis stage. The pipeline approach allows very effective use of computation time and is the basis for our software certification methodology.~~